VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Vyhodnocení vlivu metod lokálního prohledávání v evolučních algoritmech

Evaluation of the Impact of Local Search Routines in Evolutionary Algorithms

2014

David Chodúr

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **David Chodúr**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Vyhodnocení vlivu metod lokálního prohledávání v evolučních algoritmech
Evaluation of the Impact of Local Search Routines in Evolutionary Algorithms

Zásady pro vypracování:

Cílem této práce bude vyhodnocení vlivu technik lokálního prohledávání, jakými jsou např. k-OPT nebo NEH heuristic, na výkon evolučních algoritmů. Hlavní oblastí aplikací evolučních algoritmů budou kombinatorické optimalizační problémy.

Seznam doporučené odborné literatury:

[1] E. Aarts and J. Lenstra. 2003. Local Search in Combinatorial Optimization. Princeton University Press. US. ISBN: 978-0691115221
[2] M. Pinedo, 2012. Scheduling: Theory, Algorithms, and Systems. Springer, ISBN: 978-1461419860
[3] G. Onwubolu and D. Davendra. Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization. Springer, Germany, 2009. ISBN: 978-3540921509

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **MSc. Donald David Davendra, Ph.D.**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května , 2014 .....

Tímto bych chtěl poděkovat svému vedoucímu práce doc. MSc. Donaldu Davidu Davendrovi, Ph.D. za tpělivé vedení při tvorbě této práce a za představení metaheuristiky, které bych se rád věnoval i v budoucnu. Mé drahé kamarádce Ivě, která zajistila gramatickou korektnost této práce. A nakonec bych chtěl poděkovat své přítelkyni, která mne povzbuzovala a stála při mne po celou dobu tvorby.

Abstrakt

Cílem této práce je analýza vlivu lokálního vyhledávání na evoluční algoritmy. Tři diskrétné algoritmy Artificial Bee Colony, Differential Evolution a Harmony Search byly testovány s implementovaným lokálním vyhledáváním a vyzkoušeny na problému flow shop s blokováním. Testy byly provedeny na Taillardových datových souborech pro prvních 90 instancí. Z t-test analýzy algoritmů, lze vyvodit, že lokální vyhledávání značně zlepšuje genetické algoritmy.

Klíčová slova: Harmony search, Differential evolution, Artificial bee colony, Metaheuristika, lokální vyhledávání

Abstract

The aim of the thesis is to analyse the impact of local search routines on evolutionary algorithms. Three discrete algorithms of Artificial Bee Colony, Differential Evolution and Harmony Search were analysed with embedded local search routines and tested on the flow shop with blocking problem. The tests were conducted on the benchmark data sets of Taillard for the first 90 instances. From the t-test analysis on the algorithms, it can be concluded that for all the algorithms, local search improves the generic algorithms significantly.

Keywords: Harmony search, Differential evolution, Artificial bee colony, Metaheuristiks, local search

Obsah

1 Úvod

V roce 1950 byla publikována práce , jenž navrhovala postupy při produkčním plánování, během nichž není potřeba testovat všechna možná řešení, ale jen jejich omezenou část. Tyto postupy jsou dnes známy jako heuristika. Nacházejí své uplatnění v průmyslové výrobě, dopravě a dalších odvětvích. Řešené problémy jsou reprezentovány konečným množstvím řešení. Teoreticky je možné simulovat všechny možnosti těchto problémů, což je ovšem jak technicky, tak časově náročné.

Přestože heuristika vedla ke zrychlení hledání řešení, za posledních více než padesát let bylo provedeno mnoho výzkumů v tomto oboru a postupně se vyvinula Metaheuristika, která nenachází ideální, ale přípustné řešení. Byť tato metoda vedla ke zrychlení algoritmů a kvality výsledku v omezeném čase, je zde stále prostor ke zlepšení. Většina metahurestických algoritmů trpí uváznutím v lokáním maximu, čímž se neprozkoumanjí byť na první pohled nepravděpoboná, ale ve výsledku lepší řešení.

Zde tedy přichází na řadu lokální vyhledávání, jenž by mělo omezit pravděpodobnost uváznutí v lokálním maximu a mělo by se více blížit globálnímu maximu. Používá se běžně v matematice ,bioinformatice a průmyslu. Používá se samostatně jako jedna z metod metaheuristiky a využívá tzv. sousedních řešení.

Při rozšíření jiných metaheurestických metod o lokální vyhledávání by tedy mělo dojít ke zlepšení nalezených řešení. Lokální vyhledávání tentokrát ovšem nebude používáno samostatně, ale bude používáno pro korekci zkušebních řešení generovaných jinými metaheurestickými algoritmy a na toto se bude zaměřovat tato bakalářská práce.

2 Metaheuristika

2.1 Co je to metaheuristika

Jako nejjednoduší řešení pro blocking flow shop se zdá být spočítání všech možných řešení. Takovéto plýtvání zdroji je však nežádoucí a jak se ukázalo také zbytečné. Základy metaheuristiky nám napovídají, že není potřeba prohledávat veškerá možná řešení, ale je potřeba prozkoumat jen omezené množství těchto řešení a i přesto získat výsledek blížící se globálnímu maximu v omezeném čase a s omezenými zdroji. Jedná se tedy o aproximační algoritmy, jenž vycházejí z heuristiky, které se zaměřují na ekfektivní prohledání sady možných řešení. Termín metaheuristika byl poprvé představen v roce 1986. Metaheuristika byla popsána jako iterativní strategie vyhledávání, která vede vyhledávácí proces napříč možnými řešeními s cílem najít optimální řešení. Metaheuristika ovšem negarantuje nalezení globálního optima. Většina metaheurestických algoritmů se inspiruje přirozenými procesy v přírodě, jako je například Artificial bee colony ABC , Genetická evoluce GE. Nebo také v průmyslových procesech jako je simulované žíhání.

Až do nedávna neexistovala přesná definice co to vlastně metaheuristika je. Dnes bychom mohli definici shrnout jako

- Je to strategie navigující proces vyhledávání
- Cílem je nalézt řešení blížící se globálnímu optimu
- Metaheuristika není závislá na řešeném problému
- Většina metaheurestických algoritmů je nedeterministká [5]

Jak bylo také prokázáno, metaheuristické algoritmy dosahují lepších výsledků než heuristické [1]

2.2 Použití

Metaheurestické algoritmy jsou ideální pro kombinatorické problémy, kde vyhledávání probíhá nad diskrétními čísly. U většiny těchto problémů totiž náročnost roste rychleji než exponenciálně a prohledávání všech možných řešení by zabralo neúměrně mnoho času. Často se využívají například při optimalizacích výrobních linek a k celkovému plánování výroby. Je však možno je použít prakticky ve všech okruzích, ve kterých je potřeba nalézt nejlepší řešení nad konečným počtem možností.

3 Blocking flowshop

3.1 Úvod pro blocking flowshop

Blocking flowshop je jedním z nejřešenějších strojních diskrétních problémů. Skládá se z jednotlivých prací, jenž mohou znázorňovat libovolnou aktivitu a strojů, jenž zase reprezentují použitelné zdroje. Může například reprezentovat klasickou výrobní linku, kde jednotlivé práce zobrazují aktivity a stroje. Tímto způsobem lze reprezentovat až čtvrtinu průmyslových systémů, výrobních linek. Problém který se tedy u blocking flowshop snažíme řešit, spočívá v takovém seřazení permutace tak, aby byly minimanimalizovány časy, jenž musí jednotlivé práce čekat, než přejdou na následující stroj.

Problém „Blocking flowshop“ lze v diskrétní matematice zobrazit jako permutaci π skládajíci se ze sekvence prací. Nachází se zde množina prací $J = \{1, 2, 3, 4, \ldots, n\}$, kde $j \in J$ a sekvence storojů $M = \{1, 2, 3, 4, \ldots, m\}$. Každá z těchto prací bude sekvenčně zpracována na stroji $1, 2, 3, 4, \ldots, m$. Jednotlivé operace na stroji jsou zapsány jako operace $o_{j,k}$, která koresponduje s prací $j \in J$ na stroji $k = \{k = 1, 2, 3, \ldots, m\}$. Tato operace je provedena v nepřerušeném časovém úseku $p_{j,k}$. V danou chvíli může být na každém stroji právě jedna práce a každá práce může být být prováděna právě pouze na jednom stroji. Práce může opustit aktuální stroj, pouze pokud je následující stroj volný. Toto může reprezentovat například výrobní linku bez možnosti uskladnění během jednotlivých operací na stroji.

Naším cílem tedy je taková sekvence prací $j \in J$, které jsou prováděny ve stejném pořadí na m strojích tak, aby celková doba zpracováni všech operací $o_{j,k}$ byla co nejmenší.

3.2 Matematické vyjádření

Nechť permutace prací $\pi\{1, 2, 3, \ldots, n\}$ reprezentuje sekvenční plán pro zpracování jednotilivých prací a $e_{j,k}$ reprezentuje čas spuštění operace $o_{j,k}$ jenž je ilustrován na obrázku 1.

Dle Ronconil [2] lze jednotilivé časy spuštění operací spočítat následovně:

$$e_{0,1} = 0 \tag{1}$$

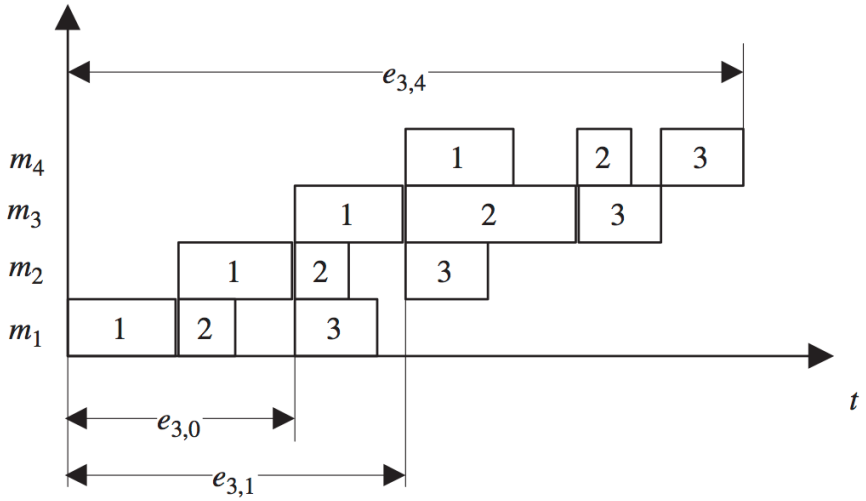
$$e_{1,k} = e_{1,k-1} + p_{1,k} \qquad k = 1, \ldots, m-1 \tag{2}$$

$$e_{j,0} = e_{j-1,1} \qquad j = 2, \ldots, n \tag{3}$$

$$e_{j,k} = max\{e_{j,k-1} + p_{j,k}, e_{j-1,k+1}\}\text{‘} \qquad j = 2, \ldots, n \quad k = 1, \ldots, m-1 \tag{4}$$

kde $e_{j,0} \quad J = 1, 2, 3, \ldots$ znázorňuje čas zahájení práce π_j na prvním stroji. Dle výpočtu popsaných výše, je vždy nejdříve pro danou práci spočítán čas na prvním stroji a následně na každém následujícím, dokud nejsou známy všechny časy prací na strojích. Následně je tedy časový rozsah dané permutace prací $\pi\{\pi_1, \pi_2, \pi_3, \ldots, \pi_n\}$ dán pomocí $C_{max} = e_{n,m}$

Následně je potřeba spočítat tzv. „tail“ neboli ocas, jenž je znázorněn jakožto $f_{j,k}$. Tail nám určuje dobu mezi posledním naložením operace $o_{j,k} \quad (k = m, \ldots, 1)$ a ukončením této operace. $f_{j,m+1}$ nám tedy znázorňuje dobu provádění mezi poslední operací $o_{j,m}$ a konečnému času provedení na posledním stroji. viz 2. Výpočet f lze zobrazit následovně



Obrázek 1: Výpočet $e_{j,k}$

$$f_{n,m+1} = 0 \quad (5)$$

$$f_{n,k} = f_{n,k+1} + p_{n,k} \qquad k = m, \ldots, 2 \quad (6)$$

$$f_{j,m+1} = fj + 1, m`qquadj = n-1, \ldots, 1` \quad (7)$$

$$f_{j,k} = max(f_{j,k+1}p_{j,k})` \qquad j = n-1, \ldots, 1 \quad k = m, \ldots, 2f_{j,1} = f_{j,2} + p_{j,1}, j+n, \ldots, 1 \quad (8)$$

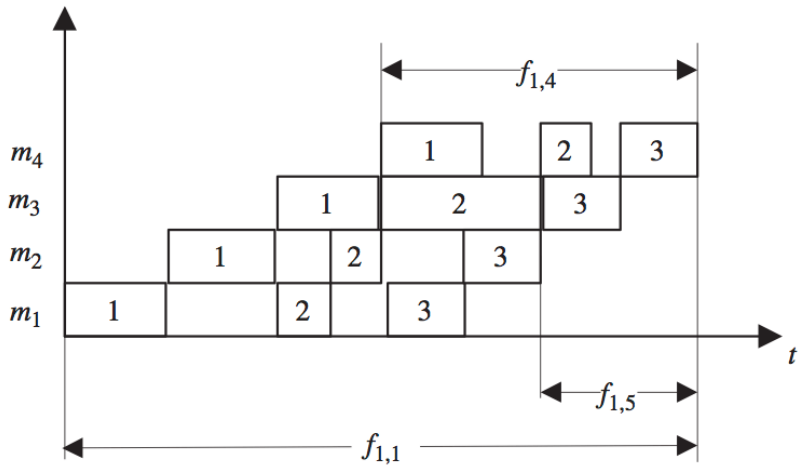
Dle výše uvedených výpočtů jsou nejdříve vypočteny ocasy poslední práce na každém stroji, poté předposlední práce, až do první práce. Následně tedy alternativní výpočet pro časový rozsah zpracování permutace prací $\pi\{\pi_1, \pi_2, \pi_3, \ldots, \pi_n\}$ je dán pomocí $C_{max} = f_{1,1}$

Naším cílem je tedy najít takovou permutaci π^*, pro kterou bude platit

$$C_{max}(\pi^*) \leq C_{max}(\pi), \quad \forall \pi \in \prod$$

3.3 Dosavadní pokusy o řešení

Problém blocking flow shop byl v posledních letech poměrně často studován, vzhledem k případech ve výrobě kde musí jednotlivé výrobky zůstavají na strojích vzhledem k nemožnosti jejich dočasného uskladnění. [3]. V tomto případě tedy mezi jednotilivými stroji neexistují žádné zásobníky a je tedy vynuceno co nejkratší navazování jednotlivých operací. Využitím možností řešení tohoto problému lze získat maximální možný pracovní



Obrázek 2: Výpočet $f_{j,k}$

čas z jednotlivých strojů. Přesto ovšem blocking flowshop nepřitáhnul příliš mnoho pozornosti, byť v minulosti zde byli pokusy o efektivní řešení tohoto problému [5]. Až v roce 2001 byl publikován článek, který navrhoval zlepšení kvality výsledku při řešení problému s vysokým počtem prací, pomocí metaheurestického genetického algoritmu [6] ,jenž pracoval s jednotlivými sekvencemi prací jako s chromozomy. Tato práce byla následně rozšířena také o křížení a lokální vyhledávání čímž vzniknul hybridní genetický algoritmus (HGA). Grabowsky dále také vyvinul taboo search (TS), kde je zamezeno zacyklení a snaží se také řešit uváznutí v lokálním optimu. Později byl navržen algoritmus diskrétní diferenciální evoluce (DDE) jenž bude studován i v této práci . Bylo zároveň prokázáno, že DDE dosahuje lepších výsledků, než swarm optimalization. Všechny tyto metody pracují na základě náhodně generovaného počátečního stavu. Jak ovšem prokázal Nawaz v roce 1983, je možno dosáhnout lepšího počátečního stavu pomocí využití NEH heuristiky.

4 Lokální vyhledávání

Metaheurestické algoritmy trpí uváznutím v lokálním maximu, a nejsou schopny se pomocí prozkoumání svého okolí z něj dostat do globálního maxima. Pro generování nových řešení sousedního řešení [4] se běžně používají operátory, záměny (swap), vložení (insert) a obrátit (inverse). Pro lokální vyhledávání navrhované v této práci je pro nás nejvhodnější operátor vložit. Lokální vyhledávání by mělo mít za následek získání lepších výsledků v menším počtu iterací, čili redukci potřebného času využití CPU.

Lokální vyhledávání lze použít během běhu algoritmu v různých situacích. [14]. V této práci budeme testovat lokální vyhledávání na všechny nové prvky diferenciální evoluce, nové permutace generované vyhledáváním harmonie a na prvky generované pomocí emploed bee v artificial bee colonny algoritmu.

4.1 Popis průběhu lokálního vyhledávání

Mějme permutaci $\pi = \{\pi_1, \pi, 2, \pi_3, \ldots, \pi_n\}$, jenž reprezentuje permutaci n prací na m strojích a náhodně vytvoříme permutaci β, jenž bude sloužit jako referenční pořadí jednotlivých testovaných prací. Jednotlivé referenční práce následně budeme postupně vkládat všechny jejich možné pozice v permutaci. Stručně řečeno, jednotlivé stochastické algoritmy nám dodávají pro lokální vyhledávání počáteční permutace, což nám umožňuje prozkoumat i jiná řešení, než které bychom získali jen vyhledáváním v sousedních řešeních.

Proceduru lokálního vyhledávání lze zapsat následovně

1. inicializujeme proměnné $i = 0$, $j = 0$ a $\beta = \{\beta_1, \beta_3, \beta_3, \ldots, \beta_n\}$ kde n je rovno počtu prací a permututaci na niž bude prováděno lokální vyhledávání π
2. nastavení $i = i + 1$ Pokud je $i > n$ pak $i = i - n$
3. odstranění práce β_i z permutace π její následné vložení na všechny pozice permutace. Získanou permutaci označíme jako π_{new}
4. Pokud je $f(\pi_{new}) > f(\pi)$, kde f představuje objektivní funkci, pak $j = 0$ a $\pi = \pi_{new}$ a pokračuj krokem 2. Pokud je $f(\pi_{new}) < f(\pi)$ pak $j = j + 1$
5. Pokud je $j <= n$ pokračuj krokem 2. Pokud $j > n$ ukončíme lokální vyhledávání.

5 Differenciální evoluce

V roce 1995 byla publikována práce Rainer Storn a Kenneth Price jenž představila tehdy nový způsob řešení problémů v diskrétním prostoru řešení. Jednalo se o metaheurestickou medotu s názvem Differenciální evoluce. Bylo prokázáno že tato metoda konverguje rychleji než simulované žíhání, jenž se těší velké oblibě jakožto velmi efektivní druh algoritmu pro řešení problému v nepřetržitém prostoru. Diferenciální evoluce patří do kategorie metaheurestických algoritmů založených na populaci. Jejím cílem tedy je najít globální optimum v konečném prostoru řešení. Skládá se ze čtyř kroků, z něchž 3 se cyklicky opakují v podstatě ve stejné formě jako je vidíme v běžné evoluci. Vytvoření základní populace, mutace, křížení a selekce. Generace je symbolizována pomocí množiny X_i^t, kde nám t určuje iterace a $i = 0, 1, 2, \ldots, PS$, jenž odkazuje na konkrétní prvek v generaci a kde iterace 0 určuje náhodně vytvořenou inicializovanou generaci. Vzhledem k tomu že klasické diferenciální evoluce pracuje s čísli s plovoucí čárkou a diskrétními čísli, budeme používat varici Diskrétní diferenciální evoluce.

5.1 Vytvoření pořáteční populace

Populace je na začátku vytvořena zcela náhodně, aby byla zajištěna co největší rozlišnost v populaci a omezila se tímto pravděpodobnost uváznutí v lokálním maximu ,což je známý problém většiny algoritmů v oboru metaheuristiky. Na začátku tedy definujeme velikost populace PS (Population size). Na základě toho vytvoříme počáteční generaci jednotlivců, jenž reprezentují jenotlivá řešení. Tímto definujeme první generaci X_a^1, kde nastavíme jako nejlepší řešení prvek X_1^1 tedy řešení první generace na první pozici. Pro generování počáteční populace je použit vzorec 9

$$X_{i,j}^0 + r \times (U_j - L_j) \tag{9}$$

kde r je náhodné rovnoměrně rozležené číslo v rozsahu od 0 do 1. U_j a L_j nám zde jen určují horní a dolní hranici rozsahu řešení.

5.2 Mutace

Mutace zajišťuje diverzitu a zvyšuje pravděpodobnost uniknutí z lokálního optima. Na začátku je nastaveno $t = t + 1$. Jednotlivá řešení jsou uloženy jako V_i^t. Pro prvky $i = 1, 2, 3, \ldots, PS$ bude tedy provedena mutace kde je vzorec dán následovně:

$$V_i^t = X_a^{t-1} + Z \times (X_b^{t-1} + X_c^{t-1})$$

kde a,b a c jsou náhodně generována čísla z rozsahu 1 až PS a jednotlivá nádná čísla jsou rozdílná, $i = 1, 2, 3 \ldots, PS$ a náhodné číslo $Z \in [0, 1]$, jenž představuje mutační faktor. Tento vzorec obsahuje 2 hlavní části, vážený rozdíl mezi dvěmi řešeními v generaci a generování mutovaného řešení. Vážený rozdíl mezi dvěma řešeními probíhá jednoduše tak, že pro každý prvek je provedena následující rovníce 10, 11

$$\Delta_i^t = Z \times (X_b^{t-1} + X_c^{t-1}) \tag{10}$$

$$\delta_{ij}^{t} = \begin{cases} x_b^{t-1} - x_c^{t-1} & if \quad rand() < Z \\ 0 & else \end{cases} \tag{11}$$

V této rovnici Δ reprezentuje vážený rozdíl mezi dvěmi řešeními a δ konkrétní pozici v δ. Pro snadnější pochopení je postup zobrazen na fig 3 V druhé části tedy použijeme náš vypočítaný vážený rozdíl Δ_i^t a přičteme k němu prvek X_a^{t-1}, čímž získáme mutovaný prvek V_i^t

$$V_i^t = X_a^{t-1} + \Delta_i^t \tag{12}$$

$$v_{ij}^t = x_{aj}^{t-1} + \delta_{ij}^t = mod((x_{aj}^{t-1} + \delta_{ij}^t + n), n) \tag{13}$$

Rovnice 13 nám popisuje výpočet jednotlivých prvků mutovovaného jedince, kde je potřeba implamentovat funkci mod, která nám vrací zbytek po dělení, kdy je první operand vydělen druhým. Díky tomu zíkáme vždy jen prvky, jenž mohou reprezentovat jednolivé práce v každé z dimenzí V_i^t. Pro lepší porozumění je celý postup zobrazen na obrázku. Mutace ovšem negeneruje použitelnou permutaci π, vzhledem k tomu, že se zde mohou opakovat jednotlivé prvky několikrát a jiné zase zcela chybět. Toto řeší až následné křížení, jenž je popsáno v další části této práce. Mutace tedy není používána ke generování použitelné permutace, ale slouží k zamezení předčasné konvergenci prvků směrem k lokálnímu maximu, jenž není zároveň globálním maximem.

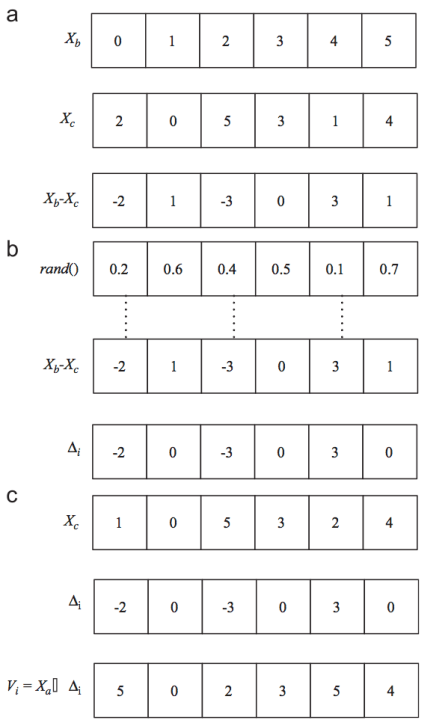
5.3 Křížení

Výsledkem křížení je již finální permutace π, jenž bude následně porovnána s korespondující permutací X_i^t. Řešení vygenerováno pomocí křížení je označeno jako U_i^t. Permutace U_i^t je generována kombinací prvku z mutované generace prvků U_i^t a prvků z generace řešení X_i^{t-1}. Základním předpokladem křížení je, že výsledná nová permutace U_i^t bude lepší než prvek v generaci řešení X_i^{t-1}. Abychom tohoto dosáhli, zvolíme tedy s pravděpoboností CR neboli crossover rate unikátní prvky mutovaného prvku V_i^t .

Vybereme unikátní prvky, jenž byli zvoleny s pravděpodovností CR. Každý prvek,který vybíráme je unikátní. Získanou množinu testovaných prvků aplikujeme na prvek X_i^{t-1}. Toto porovedeme tak, že každý prvek jeden za druhým vybereme a postupně. Vybraný testovací prvek odstraníme z X_i^{t-1} a následně jej vložíme na všechny možné pozice v permutaci. Permutaci s nejlepším možným výsledkem poté uložíme. Tento proces budeme opakovat, dokud nevyčerpáme všechny možné prvky množiny vybraných prvků na základě mutace a pravděpodobnosti křížení. Získanou permutaci následně zachováme.

5.4 Selekce

Finální částí je selekce. V této části porovnáme prvek X_i^{t-1} s naším zkušebním prvkem U_i^t. Pokud je prvek U_i^t vhodnější pro naše řešení tak se stává členem nové generace X_i^t.



Obrázek 3: Výpočet váženého rozdílu

Pokud ne ponechává se původní prvek pro příští generaci viz 14

$$X_i^t = \begin{cases} U_i^t & iff(U_i^t) \leq f(X_i^{t-1}) \\ X_i^{t-1} & else \end{cases} \tag{14}$$

Kde f znázorňuje objektivní funkci, jenž určí vhodnější prvek.

Finálním krokem je aktualizace nejlepšího doposud získaného řešení. Kroky mutace, křížení a selekce budou opakovány do té doby, než bude zajištěno dostatečně kvalitní řešení, nebo neproběhnul maximální počet iterací.

5.5 Průběh diferenciální evoluce

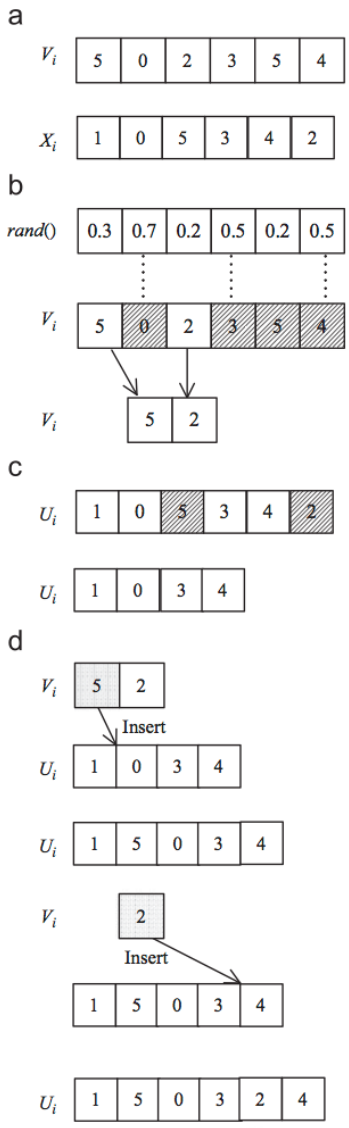
Diferenciální evoluce probíhá v následujících krocích

1. Nastavení parametrů $CR, Z, PS,$
2. Inicializace. Vygenerování populace X^0 o velikosti PS
3. Mutace.Vygenerování mutované generace V^t o velikosti PS
4. Křížení. Aplikování křížení na mutavanou generaci, čímž získáme PS možných permutací U^t
5. Selekce. Porovnání PS jedinců pro následující generaci
6. Aktualizace nejlepšího nalezeného řešení
7. Pokud byla spněna ukončovací podmínka, poslat na výstup nejlepší nalezené řešení a ukončit algoritmus. Pokud podmínka splněna nebyla pokračuj krokem 3

5.6 Rozšíření o lokální vyhledávání

Lokální vyhledávání popsáno výše , v této práci aplikujeme na permutaci U^t, s pravděpodobností P.Vzhledem k tomu, že jak diferenciální evoluce tak lokální vyhledávání dosahují dobrých výsledků v průzkumu možných řešení, očekáváme získání kvalitních výsledků pro řešení blocking flowshopu. Upravený algoritmus tedy vapadá takto :

1. Nastavení parametrů CR, Z, PS, P.
2. Inicializace. Vygenerování populace X^0 o velikosti PS.
3. Mutace.Vygenerování mutované generace V^t o velikosti PS.
4. Křížení. Aplikování křížení na mutavanou generaci, čímž získáme PS kandidátů řešení U^t.
5. Lokální vyhledávání. Aplikace lokálního vyhledávání s pravděpodobností P na kandidáty U^t.



Obrázek 4: Průběh křížení

6. Selekce. Porovnání PS jedinců pro následující generaci.
7. Aktualizace nejlepšího nalezeného řešení.
8. Pokud byla spněna ukončovací podmínka, poslat na výstup nejlepší nalezené řešení a ukončit algoritmus. Pokud podmínka splněna nebyla pokračuj krokem 3.

6 Harmony search

Harmony search byl představen v roce 2005, kde byl použit pro směřování vozidel [8] a později rozšířen Wangem [19] o několik metod pro zlepšení kvality vysledků včetně implamentace NEH a NEH_WPT [20] heuristiky. Jako v Genetic evolution zde generujeme nové prvky z již testovaných v paměti pomocí porovnání. Narozdíl od Genetic evolution bylo ovšem prokázáno, že díky své jednoduchosti dosahuje Harmony search lepších výsledků [18] .Stejně jako se v metaheurestických algoritmech snažíme o napodobení přírodních, věděckých a jiných chování, tak zde se zabýváme hledáním harmonie. Celý proces by se dal znázornit jako hudebník hledající posobě jdoucí noty pro vytvoření harmonie. Začne s náhodnými notami a postupně harmonii upravuje o půl tonu, než najde takovou pomocí svého hudebního sluchu (pro nás objektivní funkcí), která bude vyhovovat jeho potřebám. Stejně jako při každém novém zahrání je harmonie upravena pomocí náhodného vybrání tonu a následné úpravě, tak i zde platí že v každé iteraci bude harmonie upravena a idálně bude nalezena nová a lepší. Celkově se hledání harmonie dá rozdělit do pěti kroků

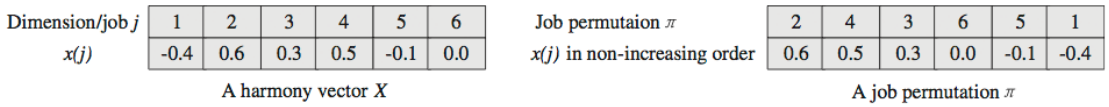
- nastavení parametrů
- počáteční naplnění paměti harmonii
- vytvoření nové harmonie
- aktualizace paměti
- kontrola podmínky pro ukončení

Základními parametry jsou velikost paměti harmonie HSM, pravděpodobnost zvážení harmonie P_{cr}, pravděpodobnost upravení harmonie P_ar, maximální velikost úpravy to ónu Db a podmínka pro ukončení. Co se týče podmínky ukončení pravděpodobně se bude jednat vždy o maximální počet improvizací nové harmonie. P_{ar} a P_{cr} slouží k nalezení lokálně a globálně zlepšených řešení. Největší váhu, zde budou mít paramtery P_{ar} a Db, jenž slouží ke generování zcela nových tónů, které pomáhají ke korigování konvergence harmonii v paměti směrem k optimálnímu řešení. Jednotlivá řešení jsou reprezentovány jako vektory $X_i\{x_i(1), x_i(2), x_i(3), \ldots, x_i(n)\}$. Tento vektor lze poté jednoduše převést na permutací prací jenž bude obsahovat jednotlivé práce 1,2,3,. . . , n , kde každá práce je unikátní, jednotlivé vektory harmonie přiřadíme tedy jednotlivým pracím na základě jejich pozice a následně seřadíme podle velikostí příslušných vektorů od největšího po nejměnší čímž získáme permutaci π resp. kadndidáta na řešení. Celý postup viz fig 5

6.1 Inicializace paměti harmonie

Na začátku je tedy potřeba vytvořit úvodní harmonie a uložit je do naší paměti $HM = \{X_1, X_2, \ldots, X_n\}$,kde X_i jsou náhodně generovanými vektory harmonie. Nechť tedy $X_i\{X_i(1), X_i(2), X_i(3), \ldots X_i(n)\}$ reprezentuje i-ty prvek z paměti harmonie HM. Lze je generovat následovně:

$$X_i(j) = U_j + r \times (U_j - L_j) \tag{15}$$



Obrázek 5: Převedení vektoru harmonie na permutaci prací

kde $j = 1, 2, \ldots, n$ a $i = 1, 2, \ldots, HMS$. U_j a L_j nám určují minimální a maximální velikost vektoru harmonie v našem případě nastaveny jako $U_j = 1$ a $L_j = -1$

Jak nám ovšem bylo prokázáno v NEH heurestikou , jenž je jedeno z nejefektivnějších řešení pro blocking flowshop, tak práce s celkově větším procesním časem, by měly mít větší prioritu než práce s celkově menším procesním časem. Zároveň ovšem u blocking flowshopu mohou práce s větším procesním časem za sebou, způsobit dlouhé čekací doby následujících prací, což může mít za následek celkový nárůst časového rozsahu všech prací. Na základě tohoto lze předpokládat zlepšení výsledků v případě seřazení prací od nejmenších po největší. Z toho poznatku také vychází NEH_WPT hereustika. Pro zkvalitnění počátečního stavu, tedy první prvek v paměti bude seřazen od největší po nejmenší a druhý naopak od nejmenšího po největší. Tuto permutaci π je ovšem za potřebí převést na vektor harmonie, čehož lze dosáhnout pomocí následujícího vzorce

$$x(\pi(j)) = U_j - \frac{U_j - L_j}{n-1} \times (j-1) \qquad j = 1, 2, 3, \ldots, n \tag{16}$$

kde U_j a L_j nám určují minimální a maximální velikost vektoru harmonie.

6.2 Improvizace nové harmonie

Stejně jako hudebník prochází jednotlivé tony a postupně je upravuje pro celkové zlepšení kvality harmonie, tak tento algoritmus postupně jemně upravuje jendolivé pozice vektorů za cílem získání lepšího výsledku. Probíhají zde 3 části algoritmu

- zvážení paměti.
- zlepšení tonu.
- náhodný výběr.

Pro každý prvek ve vektoru harmonie X_{new} tedy provedeme následující kroky. Začínáme zvážením již existujícího tonu v paměti a vygenerujeme si uniformní náhodné číslo $r_{cr} \in (0, 1)$. Pokud tady $P_{cr} > r_{cr}$ pak je náhodně vybrán z paměti harmonie X prvek X_a a $X_{new}(i) = X_a(i)$. Následně tedy s pravděpodobností P_{ar} provedeme úpravu tónu 17

$$X_{new}(j) = X_{new}(j) \pm Db \times r \qquad r \in (0, 1) \tag{17}$$

kde r je uniformní náhodné číslo a operátor $+$ nebo $-$ je zvolen náhodně.

Pokud ovšem $r_{cr} > P_{cr}$ pak je nový tón harmonie generován náhodně, stejně jako na začátku pomocí rovnice 15 .

Následujícím krokem je aktualizace paměti harmonie, kdy je zvolen nejhorší prvek z paměti HM a označíme jej jako X_w. Pokud je námi generovaná harmonie X_{new} lepší než X_w tak zaujme jeho pozici v paměti a harmonie X_w je zahozena. Dle Mahdaviho [**?**] správné nastavení P_{ar} a P_{cr} pomáhá k prozkoumání řešení blížících se lokálnímu a globálnímu optimu.

6.3 Průběh výpočtu

Pro lepší pochopení větvení je celý proces zobrazen pomocí pseudokódu.

for j=1 to n **do**
  if $P_{cr} > r_{cr}$ **then** $X_{new}(i) = X_a(i)$
    if $P_{ar} > r_{ar}$ **then** $X_{new}(j) = X_{new}(j) \pm Db \times r \qquad r \in (0, 1)$
    else$X_{new}(j) = U_j + r \times (U_j - L_j)$
    end if
  end if
end for

6.4 Průběh algoritmu

Harmony search patří k jednoduším algoritmům metaheuristiky a je poměrně nenáročný. Hledání harmonie probíhá následovně

1. Nastavení parametrů MS, P_{cr}, P_{ar}, Db.
2. Inicializace paměti harmonie o velikosti HM a nalezení nejhorší harmonie v paměti X_w.
3. Improvizace nové harmonie X_{new}.
4. Pokud je $f(X_{new}) > f(X_w)$, kde f představuje objektivní funkci, pak X_new nahradí místo X_w v paměti harmonie.
5. Pokud byla spněna ukončovací podmínka, poslat na výstup nejlepší nalezené řešení a ukončit algoritmus. Pokud podmínka splněna nebyla pokračuj krokem 3.

6.5 Rozšíření o lokální vyhledávání

Lokální vyhledávání aplikujeme na permutaci X_new s pravděpodobností P

1. Nastavení parametrů MS, P_{cr}, P_{ar}, Db.
2. Inicializace paměti harmonie o velikosti HM a nalezení nejhorší harmonie v paměti X_w.
3. Improvizace nové harmonie X_{new}.
4. Aplikace lokálního vyhledávání na prvek X_new s pravděpodobností P.

5. Pokud je $f(X_{new}) > f(X_w)$, kde f představuje objektivní funkci, pak X_new nahradí místo X_w v paměti harmonie.

6. Pokud byla spněna ukončovací podmínka, poslat na výstup nejlepší nalezené řešení a ukončit algoritmus. Pokud podmínka splněna nebyla pokračuj krokem 3.

7 Artificial bee colonny

Artificial bee colonny patří mezi rojové algoritmy. Vychází z chování například ryb, ptáků a hmyzu jako například včel, mravenců a ostatní druhy, jenž fungují nějakým způsobem v roji. Každý jedinec v roji má svou roli, bez toho aby na něj musel jiný nějakým způsobem dohlížet. Lokální pravidla pro jednotlivce bez jakékoli návaznosti na globální řízení tvoří to, čemu říkáme inteligence roje. Jednotlivé zdroje jsou přesto využívány efektivně inteligencí roje. Roj v podstatě stojí na čtyřech základních vlastnostech, jak je popsal ve své práci Karaboga [9]

- Pozitivní zpětná vazba, jenž podporuje tvorbu vhodných struktur. Toto chování je například tančení včel před úlem, když popisují nový zdroj potravy.
- Negativní zpětná vazba, jenž pomáhá stabilizaci. Určuje také například vyčerpaný zdroj potravy, konkurenci či zahlcení. V podstatě se jedná o zahození již nevhodných řešení.
- Náhodné hledání nových řešení, chyb, nebo provádění náhodných činností. V podstatě se jedná o kreativitu intelgentního roje, jenž výrazně přispívá k získávání nových řešení.
- Celková sebe organizace vyžaduje minimální hustotu vzájemně podobných jedinců, což jim umožňuje těžit z výsledku jejich vlastních aktivit, stejně jako aktivit ostatních.

Roj by měl mít základní vlastnosti :

- roj musí být schopen zákaldních časových výpočtů,
- roj musí umět rozpoznat kvalitativní faktory, jako je například kvalita zdroje potravy ,
- zdroje by měli být rozloženy rovnoměrně do celého spektra ,
- roj musí být schopen změnit své chování pokud je potřeba a pokud se změna energeticky vyplatí, resp. je výhodná.

V posledních letech přitáhlo pozornost rojové chování v kolonii mravenců, putování ptáků a také chování ryb a byli úspěšně implamentovány v oborech robotiky, her, ekonomiky a jiných oborech. Tyto techniky byly impamentovány dříve, než chování roje včel některé z nich i o 10 let dříve. Zde konkrétně testovaný algoritmus roje včel patří mezi novější a předčí i velmi oblíbený algoritmus genetic evolution [10] a byl úspěšně použit i na jiné problémy [11]

7.1 Chování jednotilivých jedinců

V kolonii včel se běžně nachází 2 druhy včel. Nemáme zde ovšem na mysli trubce, dělnice a královnu. Zde je myšlena zaměstnaná včela jenž má již svůj zdroj potravy z nějchž získává potravu. Dalším druhem je včela jenž neustále vyhledává nový zdroj potravy.Náš

algoritmus pracuje ještě se třetím druhem a to je v dohlížejcící včela, jenž vyhledává zdroje potravy v okolí již nalezených kvalitních zdrojů. Aby tento proces mohl správně fungovat je duležitou součástí je nepochybně sdílení informací k zajištění kolektivní znalosti zdrojů potravy.

7.2 Popis algoritmu

Na začátku algoritmu je potřeba nastavit základní vlastnosti. Jedná se o limit zlepšení zdroje potravy, počet iterací a počet zdrojů potravy, jenž jsou v danou chvíli používány. Co se týče interpretace algoritmu nachází se zde mnoho různých řešení. Je zde dána jen základní interpretace, která se opakuje v mnoha podobách od zlepšení inicializačních parametrů (což je dnes běžnou praxí nejen u artificial bee colonny), samozdokonalujícím se vyhledávání v okolí [13] , vylepšením výběru ve fáži Scout bee [12] a případné zdokonalení pomocí částí z Differential evolution.

V artificial bee colonny budeme pracovat se třemi druhy včel .

- Zaměstnaná včela.
- Včela průzkumník.
- Dohližející včela.

Každá z nich má svůj úkol pro celkové zlepšení zdrojů potravy. Základními parametry Artificial bee colonny je počet zdrojů potravy SN, $limit$ a počet iterací $loop$. Limit nám zde určije po kolika iteracích je zdroj potravy zahozen, pokud nebyl v limitu iterací jakkoliv zdokonalen. V následujících částech budou všechny včely, resp. části algoritmu popsány.

7.2.1 Inicializace zdrojů

Jako u většiny populačních metaheurestických algoritmů je prvním krokem naplnění populace počátečnímy náhodnými hodnotami. Vygenerujeme tedy SN permutací X_i^t kde i určuje index permutace v populaci a t označuje iteraci.

7.3 Zaměstnaná včela / Employed bee

Tato řást algoritmu simuluje včelu dělnici. Jedná se o včelu, která má již svůj zdroj potravy nalezen a snaží se pouze nají lepší řešení v jeho blízkém okolí. Jsou zde použity operátory pro vložení a záměnu v permutaci jenž se běžně používají při pro generování sousedních řešení . Vložení probíhá vybráním náhodného prvku z permutace $X_i^t - 1$ a vložením na novou pozici. Zaměnění probíhá vybráním dvou náhodných prvků z permutace X_i^t a následně záměnou jejich pozic v permutaci. Pro zkvalitnění nalezených řešení během Employed bee fáze budeme používat 4 operátory, z nichž bude vždy jeden náhodně vybrán.

- Vlož jeden
- Zaměň jeden

- Vlož dva
- Zaměň dva

Pro každou Emploed bee v každé iteraci zvolíme právě jeden z výše zmiňovaných operátorů. Celý postup employed bee lze popsat následovně :

1. zvolení jednoho z operátorů vložení, nebo záměny viz 7.3
2. aplikováni operátoru na prvek X_i^t, čímž získáme novou permutaci X_new
3. Porovnání prvků a uložení zdroje potravy :

$$X_i^t = \begin{cases} X_{new} & iff(X_i^t) \leq f(X_{new}) \\ X_i^{t-1} & else \end{cases} \tag{18}$$

kde funkce f reprezentuje objektivní funkci, zhodnocující množství nektaru zdroje potravy.

7.4 Včela průzkumník / Scout bee

Jedná se v podstatě o rozšíření Employed bee. Mezi zdroji potravy jsou takové, jenž dosahují lepších výsledků než jiné, čímž by měli skrývat větší potenciál pro nalezení sousedního zdroje potravy. V klasickém Artificial bee algoritmu je zdroj potravy X_i vybrán s pravděpodobností P_i. Vzhledem k tomu, že se ovšem systém turnaje osvědčil v mnoha jiných metaheurestických algoritmech, včetně genetic evolution, díky jeho jednoduchosti a zvýšené šanci uniknout z lokálního optima. V našem algoritmu tedy implementujeme turnaj o dvou účastnících, jenž budou na základě objektivní funkce $f(X_i)$ posouzeny. Zdroj potravy, jenž dosáhne lepšího výsledku je následně podroben stejnému procesu jako zdroje potravy během Employed bee fáze. Pokud je získaný zdroj potravy lepší než současný, nahradí jeho místo mezi zdroji potravy a starý zdroj potravy je zahozen.

7.5 Dohlížejicí včela / Onlooker bee

Může se stát, že jistý zdroj potravy nemůže být dále zdokonalen pomocí vyhledávání v jeho sousedních řešení. To se může stát, pokud například zůstal v lokálním maximu, čemuž se snažíme vyhnout. Pokud zdroj potravy nebyl vylepšen v $limit$ iterací, je zahozen a nahrazen zcela novým zdrojem potravy. Generování nového zdroje potravy můžeme provést několika způsoby, například náhodně stejně jako jsme generovali zdroje potravy při inicializaci, případně se můžeme pokusit vylepšit zdroj nejlepší nalezený zdroj potravy. V této konrétní verzi Artificial bee colony algoritmu budeme používat pro generování nové zdroje potravy, část z diferenciální evoluce. Konkrétně tedy mutace 5.2 a křížení 5.3, čímž získáme nový prvek s dostatečnou diverzitou.

7.6 Průběh algoritmu ABC

Nyní, když jsme obeznámeni se všemi prvky ABC algoritmu, výpočetní procedura je následovná:

1. Nastavení parametrů $limit$, počet zdrojů potravu SN a počet iterací. Počet jednotlivých včel bude stejný jako počet zdrojů potravy.
2. Inicializace zdrojů potravy $X^0 = \{X_1^0, X_2^0, X_3^0, \ldots, X_{SN}^0, \}$
3. Employed bee. Během této fáze se snažíme najít sousední řešení již námých zdrojů potravy. Pro získání sousedního zdroje potravy využijeme rovnici popsanou výše 7.3
4. Scout bee. Implamentujeme systém turnaje o dvou soutěžících.

   Pro $i = 1, 2, 3, \ldots, PS$ provedeme následující kroky

   (a) Zvolíme zdroje potravy X_a^t a X_b^t, kde a, b jsou náhodně generována čísla v rozsahu $(0, PS)$

   (b) Zdroj potravy s větším množstvím nektaru podrobíme procesu vložení a záměny popsané výše 7.3 , čímž získáme prvek X_{new}

   (c) Pokud je nový prvek lepší než námi vybraný prvek v turnaji, pak nový prvek nahradí jeho pozici mezi zdroji potravy a starý prvek je zahozen.
5. Scout Bee. Každý prvek, jenž nebyl vylepšen v $limit$ pokusu zlepšení je zahozen a nahrazen novým, jenž se získá pomocí mutace a křížení z aktuální generace.
6. Aktualizuj nejlepší nalezené řešení
7. Pokud je splněna podmínka ukončení pošli nejlepší nalezené řešení na výstup a ukonči, jinak pokračuj krokem 3.

7.7 Rozšíření ABC algoritmu o lokální vyhledávání

Lokální vyhledávání v tomto případě nebudeme aplikovat na všechny generované zdroje potravy (možná řešení), ale použe pokud employed bee nejde lepší řešení. Toto řešení má vysokou pravděpodobnost, že může být efektivně dále zdokonaleno pomocí lokálního vyhledávání.

1. Nastavení parametrů $limit$, počet zdrojů potravu SN a počet iterací. Počet jednotlivých včech bude stejný jako počet zdrojů potravy.
2. Inicializace zdrojů potravy $X^0 = \{X_1^0, X_2^0, X_3^0, \ldots, X_{SN}^0, \}$.
3. Fáze onlooker bee kde pro $i = 1, 2, 3, \ldots, PS$ provedeme následující kroky.

   (a) zvolení jednoho z operátorů vložení nebo záměny viz 7.3.

(b) aplikováni operátoru na prvek X_i^t, čímž získáme novou permutaci X_new.

(c) porovnání prvků a uložení zdroje potravy :

$$X_i^t = \begin{cases} X_{new} & iff(X_i^t) \leq f(X_{new}) \\ X_i^{t-1} & else \end{cases} \tag{19}$$

kde funkce f reprezentuje objektivní funkci, zhodnocující množství nektaru zdroje potravy.

4. Fáze scout bee, kde implamentujeme systém turnaje o dvou soutěžících.

   Pro $i = 1, 2, 3, \ldots, PS$ provedeme následující kroky

   (a) Zvolíme zdroje potravy X_a^t a X_b^t kde a, b jsou náhodně generována čísla v rozsahu $(0, PS)$

   (b) Zdroj potravy s větším množstvím nektaru podrpobíme procesu vložení a záměny popsané výše 7.3 , čímž získáme prvek X_{new}

   (c) Pokud je nový prvek lepší než námi vybraný prvek v turnaji, pak nový prvek nahradí jeho pozici mezi zdroji potravy a starý prvek je zahozen.

5. Scout Bee. Každý prvek, jenž nebyl vylepšen v $limit$ pokusu zlepšení je zahozen a nahrazen novým, jenž se získá pomocí mutace a křížení z aktuální generace.

6. Aktualizuj nejlepší nalezené řešení

7. Pokud je splněna podmínka ukončení, pošli nejlepší nalezené řešení na výstup a ukonči, jinak pokračuj krokem 3.

8 Testování

8.1 Tailradovo měřítko výkonosti

Pro naš experiment budeme používat sadu problémů pro blocking flowshop navržený Taillardem [15], jenž se skládá z 12ti ruzných problémů od 20ti prací na 5ti strojích až po 500 prací na 20 strojích. Každý z nich obsajuje 10 unikátních instancí.

Pro každý z testů vypočítáme procentuální realtivní rozdíl (PRD), jenž lze vypočítat následovně $PRD = 100 \times \frac{(C^{RON} - C^a)}{C^{RON}}$, kde C^{RON} představuje dobu potřebnou ke zpracování všech prací na všech strojích vypočítnou Ronconi, použitou v práci [16], z nichž jsme přejali tyto hodnoty. C^a představuje námi porovnávanou dobu potřebnou ke zpracování všech prací na všech strojích.

8.2 Způsob porovnávání

Námi vytvořené algoritmy budeme porovnávat pomocí PRD na problémech vytvořených Tailardem v rozmezí 20 prací na 5ti strojích až po 100 prací na 10ti strojích. Jednotlivé algoritmy poté mezi sebou porovnáme v tabulkách a to jak jejich získané hodnoty, tak čas potřebný k provedení výpočtů. Každý problém bude prováděn v 5ti nezávislých iteracích. Porovnávané hodnoty budou PRD, maximální PRD, minimální PRD, průměrné PRD a standartní deviace SD. Existuje mnoho problémů, na kterých lze testovat metaheuristické algoritmy. Pro naše potřeby je vyhovujicí blocking flowshop, jenž byl popsán výše v této práci. Jednotlivé algoritmy probíhaly na úlohách vytvořených Tailardem. Vzhledem k velké výpočtové náročnosti budeme testovat nami vytvořené algoritmy jen na problémech v rozmezí 20 prací na 5ti strojích až po 100 prací na 10ti strojích. Jako stroj pro testování byl použit notebook Asus K50AB s dvoujádrovým procesorem AMD Athlon 2.0GHz a 3GB RAM. Jako programovací jazyk byl použit C++ a výpočty probíhali pod OS Windows 8 32bit.

9 Testování difereneciální evoluce

9.1 Nastavení parametrů

Nastavení parametrů diferenciální evoluce bylo následující:

- Velikost populace $PS = 100$.
- Počet iterací byl nastaven na 50.
- Pravděpodobnost mutace $Z = 0.7$.
- Pravděpodobnost křížení $CR = 0.7$.
- Pravděpodonost provedení lokálního vyhledávání $P = 0.2$.

Lokální vyhledávání bylo aplikováno s pravděpodobností P na nové jedince, jenž byli vytvořeni ve fázi křížení.

9.2 Porovnání výsledků

Již na první pohled z tabulek 1 a 2 je zjevné, že výsledky diferenciální evoluce s lokálním vyhledáváním dosahují lepších hodnot a přibližují se tedy více ke globálnímu optimu. Samotná diferenciální evoluce nebyla většinou schopna předčit Ronconiho výsledky. Pouze některé výsledky byli toho schopny. Oproti tomu po obohacení o lokální vyhledávání veškeré průměrné hodnoty překonaly Ronconiho a to většinou včetně nejhorších výsledků testů. Co se týče nejlepších výsledků, byly schopny překonat Ronconiho i o více než 7% Lokální vyhledávání má ovšem nevýhodu ve své časové náročnosti, jenž je větší přibližně troj až čtyřnásobně. Differenciální evoluce tedy velmi dobře zajišťuje dobré startovní podmínky pro lokální vyhledávání a tato kombinace výrazně pomáhá k úniku z lokálního maxima a k prozkoumání širšího spektra jednolitvých řešení a konverguje tedy rychleji.

Tabulka 1: Výsledky diferenciální evoluce.

JxM	APRD	MinPRD	MaxPRD	SD	T(s)
20x5	−3.95	−5.80	0.15	0.79	2.00
20x10	−1.77	−4.71	2.47	1.49	3.35
20x20	0.98	−1.46	3.77	0.83	5.57
50x5	−6.11	−10.12	−2.06	1.18	21.01
50x10	−3.36	−6.00	−1.08	1.02	39.93
50x20	−1.52	−4.73	2.27	1.60	76.17
100x5	−12.35	−14.83	−8.81	1.03	154.65
100x10	−6.07	−8.85	−3.11	1.17	307.69
100x20	−4.08	−5.66	−1.90	0.83	661.10

Tabulka 2: Výsledky diferenciální evoluce s lokálním vyhledáváním.

JxM	APRD	MinPRD	MaxPRD	SD	T(s)
20x5	0.21	−0.55	1.61	0.39	4.15
20x10	1.69	−0.24	5.36	1.46	8.02
20x20	3.58	2.13	6.27	0.82	15.39
50x5	2.96	0.31	4.75	1.00	66.69
50x10	4.41	2.73	6.20	0.83	134.71
50x20	4.84	2.63	7.28	1.40	271.59
100x5	0.51	−0.97	2.30	0.64	588.46
100x10	4.25	3.19	6.97	0.67	1263.22
100x20	3.72	2.31	4.62	0.57	2580.77

Tabulka 3: Párový t-test pro DE proti DE+LS

Set	H_0	H_1	t-value	p-value	$p < 0.05$	H_0	H_1
20×5	DE	DE + LS	32.9090	0.0001	ano	reject	accept
20×10	DE	DE + LS	34.5821	0.0015	ano	reject	accept
20×20	DE	DE + LS	29.5359	0.0001	ano	reject	accept
50×5	DE	DE + LS	53.3565	0.0001	ano	reject	accept
50×10	DE	DE + LS	69.1734	0.0001	ano	reject	accept
50×20	DE	DE + LS	70.1624	0.0001	ano	reject	accept
100×5	DE	DE + LS	86.7576	0.0001	ano	reject	accept
100×10	DE	DE + LS	66.2047	0.0001	ano	reject	accept
100×20	DE	DE + LS	50.4590	0.0001	ano	reject	accept

10 Porovnání Harmony search

10.1 Nastavení parametrů Harmony search

Nastavení Harmony search bylo následující :

- Nastavení horní a spodní hranice harmonie při generování inicializační paměti bylo v rozsahu $(-1, 1)$.
- Velikost paměti $MS = 5$ dle Mahdavi [17].
- Počet iterací byl nastaven na 2000.
- Parametry $P_{cr} = 0.75$, $P_{ar} = 0.25$ a $Db = 0.25$.
- Pravděpodonost provedení lokálního vyhledávání $P = 0.2$.

Lokální vyhledávání zde bylo s pravděpodoností P aplikováno na všechny nově získané harmonie, jenž byly vytvořeny při improvizaci.

10.2 Porovnání výsledků

Harmony search měl dobu provedení 2000 iterací pod 0.00 vteřin. Tento algoritmus není populačně založen, což znamená že počet testovaných jedinců je roven počtu iterací. Harmony search se tedy prokázal jako nenáročný. Jak ovšem můžeme vidět v tabulce 4 velmi pomalu konverguje a v takovémto počtu iterací nenalezl vhodná řešení, s nejhorším výsledkem dosahujícího $-22.22\%PRD$.

Po implementaci lokálního vyhledávání došlo ovšem ke drastickému zlepšení, kdy se $APRD$ dostalo do kladných hodnot a maximální PRD dosáhlo i zlepšení o 7.18% Výpočetní náročnost ovšem také vzrostla a to u poslední sady úloh, až na 818.44 vteřin. Harmony search nám zde tedy poskytnul dobré počáteční podmínky pro provedení lokálního vyhledávání, díky čemuž dokázal algoritmus uniknout z lokálního maximu a přiblížit se maximu globálnímu.

Tabulka 4: Výsledky Harmony search.

JxM	APRD	MinPRD	MaxPRD	SD	T(s)
20x5	−13.73	−17.82	−10.35	1.69	0.00
20x10	−9.93	−17.79	−4.87	1.93	0.00
20x10	−4.60	−7.84	−2.10	1.43	0.00
50x5	−16.72	−22.22	−12.78	1.69	0.00
50x10	−11.35	−15.87	−7.75	1.45	0.00
50x20	−8.40	−12.92	−3.03	2.14	0.00
100x5	−22.27	−24.98	−18.37	1.25	0.00
100x10	−12.79	−16.03	−9.62	1.26	0.00
100x20	−9.07	−10.67	−6.87	0.73	0.00

Tabulka 5: Výsledky Harmony search s lokálním vyhledáváním.

JxM	APRD	MinPRD	MaxPRD	SD	T(s)
20x5	−0.01	−1.09	1.39	0.35	1.13
20x10	1.50	−0.57	5.36	1.60	2.17
20x10	3.41	1.88	6.27	0.79	4.05
50x5	2.47	0.12	4.35	1.00	19.76
50x10	3.88	2.38	5.87	0.87	45.52
50x20	4.69	2.52	7.06	1.46	82.19
100x5	−0.03	−1.75	2.33	0.65	185.06
100x10	3.65	2.56	5.59	0.63	427.54
100x20	3.42	1.98	4.71	0.60	818.44

Tabulka 6: Párový t-test pro HS proti HS+LS

Set	H_0	H_1	t-value	p-value	$p < 0.05$	H_0	H_1
20×5	HS	HS + LS	47.4285	0.0001	ano	reject	accept
20×10	HS	HS + LS	36.6669	0.0015	ano	reject	accept
20×20	HS	HS + LS	47.2710	0.0001	ano	reject	accept
50×5	HS	HS + LS	87.1252	0.0001	ano	reject	accept
50×10	HS	HS + LS	80.3736	0.0001	ano	reject	accept
50×20	HS	HS + LS	77.0481	0.0001	ano	reject	accept
100×5	HS	HS + LS	162.4776	0.0001	ano	reject	accept
100×10	HS	HS + LS	101.6994	0.0001	ano	reject	accept
100×20	HS	HS + LS	116.6173	0.0001	ano	reject	accept

11 Porovnání Artificial bee colony

11.1 Nastavení parametrů Artificial bee colony

Artificial bee colony je, co se týče parametrů jednoduchý. Musíme zde ovšem vzít v potaz, že jsme zároveň přejali mutaci a křížení z diferenciální evoluce. Lokální vyhledávání jsme implamentovali v části emploed bee, v případě, že bylo nalezeno lepší řešení u něhož je větší pravděpodobnost nalezení lepšího zdroje potravy pokud na něj aplikujeme lokální vyhledávání. Pro testování Artificial bee colony algoritmu jsme použili následující hodnoty :

- Pravděpodobnost mutace $Z = 0.7$.
- Pravděpodobnost křížení $CR = 0.7$.
- Počet zdrojů potravy $SN = 50$.
- Limit po kolika pokusech o zlepšení má být zdroj potravy zahozen $limit = 20$.
- Počet iterací byl nastaven na 100.

11.2 Porovnání výsledků

Z tabulky 7 lze zjistit, že přestože čas průběhu algoritmu probíhá pod 1 vteřinu, velmi se blíží výsledkům diferenciální evoluce, jenž trvá několikanásobně déle. Lze tedy říci, že tento algoritmus konverguje rychle, avšak stále zůstává v lokálním maximu a v průměru nedosahuje výsledků, jenž představil Ronconi, byť se mu některé hodnoty vyrovnali a předčili jej. Částečně má podíl na těchto výsledcích použití mutace a křížení pro nově generované prvky, pokud byl starý zdroj potravy zahozen, což umožnilo širší prozkoumání možných řešení.

Po implamentaci lokálního vyhledávání však opět došlo k výraznému zlepšení výsledků, kde ve většině případů i nejhorší nalezená hodnota překonala Ronconiho. Lokální vyhledávání tedy zajistilo široké prozkoumání prostoru možných řešení a zlepšilo celkovou konvergenci algoritmu s minimální deviací SD jednotlivých testů. Artificial bee colonny tedy podával stabilně velmi přijatelné výsledky.

Tabulka 7: Výsledky Artificial bee colony.

JxM	APRD	MinPRD	MaxPRD	SD	T(s)
20x5	−4.55	−7.48	−1.89	1.03	0.05
20x10	−2.45	−5.23	3.14	1.50	0.10
20x10	0.46	−2.00	3.21	1.05	0.18
50x5	−7.37	−10.98	−3.83	1.24	0.19
50x10	−4.45	−7.09	−2.24	0.92	0.38
50x20	−2.62	−6.02	1.26	1.75	0.85
100x5	−13.61	−16.04	−10.92	1.01	0.37
100x10	−7.19	−10.28	−4.52	0.98	0.83
100x20	−4.79	−6.06	−3.29	0.54	2.24

Tabulka 8: Výsledky Artificial bee colony s lokálním vyhledáváním.

JxM	APRD	MinPRD	MaxPRD	SD	T(s)
20x5	−0.01	−1.17	1.76	0.44	0.76
20x10	1.37	−0.66	5.36	1.49	1.50
20x10	3.31	1.92	6.07	0.75	2.63
50x5	2.43	0.21	4.15	1.01	12.37
50x10	3.82	2.33	5.45	0.85	24.47
50x20	4.45	2.12	6.71	1.41	49.54
100x5	−0.08	−2.08	2.21	0.70	108.94
100x10	3.64	2.51	6.25	0.67	246.51
100x20	3.36	1.92	4.43	0.56	476.50

Tabulka 9: Párový t-test pro ABC proti ABC+LS

Set	H_0	H_1	t-value	p-value	$p < 0.05$	H_0	H_1
20×5	ABC	ABC + LS	23.8683	0.0001	ano	reject	accept
20×10	ABC	ABC + LS	33.4568	0.0015	ano	reject	accept
20×20	ABC	ABC + LS	34.1334	0.0001	ano	reject	accept
50×5	ABC	ABC + LS	69.6325	0.0001	ano	reject	accept
50×10	ABC	ABC + LS	59.9083	0.0001	ano	reject	accept
50×20	ABC	ABC + LS	24.0680	0.0001	ano	reject	accept
100×5	ABC	ABC + LS	118.3310	0.0001	ano	reject	accept
100×10	ABC	ABC + LS	28.7635	0.0001	ano	reject	accept
100×20	ABC	ABC + LS	99.5722	0.0001	ano	reject	accept

12 Porovnání testovaných algoritmů

12.1 Bez lokálního vyhledávání

V tabulce 10 lze porovnat výsledky jednotlivých algoritmů pro všech 80 testovaných sad pro blocking flow shop. Nejhorších výsledků zde dosahuje jednoznačně Harmony search. Diferenciální evoluce dosahuje lepších výsledků než Artificial bee colony. Artificial bee colonny ovšem dokázalo tyto výsledky získat za několikanásobně kratší dobu. Lze zde velmi dobře vidět rozdíl ve výsledcích představených Ronconim a námi použitých algorimů, kde v případě úloh o 100 pracích na 10ti strojích lze vidět rozdíl až několik set bodů ve prospěch RON, byť na jednoduších úlohách jsme získávali přijatelná řešení.

12.2 S lokálním vyhledáváním

V tabulce 12 lze vidět získané výsledky algoritmů obohacených o lokální vyhledávání, pro jednotlivé úlohy. Je zde vždy zobrazen nejlepší výsledek z 5ti provedených testů. Díky lokálnímu vyhledávání, dokázaly všechny algoritmy uniknout z lokálního maxima a přiblížit se globálnímu maximu. Všechny tři algoritmy ve většině případech překonaly výsledky získané Ronconilem. Rozdíl mezi jednotlivými výsledky lze lépe pozorovat v plohách s větším počtem prací. Paradoxně ve výsledku nejméně profitoval z lokálního vyhledávání algoritmus využívající diferenciální evoluci, který bez lokálního vyhledávání dosahoval nejlepších výsledků. Naopak Harmony search pro několik úloh dokonce získal nejlepší získaná řešení, přičemž algoritmus bez lokálního vyhledávání se ukázal jako jednoznačně nejhorší. Artificial bee colony také profitavalo z lokálního vyhledávání a ze všech tří algoritmů po implementaci lokálního vyhledávání získal také nejlepší časy, které potřeboval k provedení nastaveného počtu iterací s průměrnou deviací SD 0.9% a časem provedení výpočtů více než 6krát menším než bylo zapotřebí u diferenciální evoluce.

Tabulka 10: Porovnání nejlepších výsledků testovaných algoritmů.

JxM	ABC	DDE	HS	RON	JxM	ABC	DDE	HS	RON	JxM	ABC	DDE	HS	RON
20x5					50x5					100x5				
1	1419	1431	1531	1384	31	3414	3409	3793	3151	61	7177	7024	7641	6455
2	1443	1459	1557	1411	32	3588	3569	3877	3395	62	7010	6982	7612	6214
3	1325	1348	1441	1294	33	3354	3379	3745	3184	63	6902	6782	7372	6124
4	1507	1505	1653	1448	34	3508	3422	3725	3303	64	6775	6707	7364	5976
5	1397	1364	1531	1366	35	3558	3431	3841	3272	65	7026	6876	7555	6173
6	1426	1428	1544	1363	36	3577	3585	3897	3400	66	6910	6909	7466	6094
7	1413	1445	1554	1381	37	3407	3386	3672	3228	67	7034	7063	7608	6262
8	1433	1453	1562	1384	38	3416	3405	3809	3260	68	6987	6860	7524	6061
9	1404	1431	1547	1378	39	3223	3273	3574	3104	69	7181	7111	7713	6474
10	1345	1334	1420	1283	40	3495	3467	3742	3264	70	7207	7186	7760	6366
20x20					50x10					100x10				
11	1764	1745	1897	1698	41	4037	4030	4340	3913	71	7949	7868	8381	7496
12	1879	1899	2001	1836	42	3899	3860	4117	3798	72	7838	7699	8232	7281
13	1737	1737	1810	1677	43	3874	3869	4230	3723	73	7909	7807	8273	7400
14	1571	1604	1729	1622	44	4022	4037	4341	3885	74	8144	8008	8461	7670
15	1671	1669	1832	1658	45	4022	3991	4239	3934	75	7867	7860	8299	7317
16	1661	1638	1741	1640	46	4015	3967	4275	3831	76	7653	7652	8122	7301
17	1668	1669	1757	1634	47	4076	4087	4320	3957	77	7904	7881	8280	7247
18	1796	1799	1880	1741	48	3938	3896	4145	3774	78	7738	7713	8159	7315
19	1804	1813	1899	1777	49	3989	3917	4247	3784	79	7976	7877	8365	7631
20	1840	1829	1937	1847	50	4032	4044	4272	3928	80	7938	7858	8349	7411

Tabulka 11: Porovnání nejlepších výsledků testovaných algoritmů. Pokračování tabulky 10.

JxM	ABC	DDE	HS	RON	JxM	ABC	DDE	HS	RON	JxM	ABC	DDE	HS	RON
20x20					50x20					100x20				
21	2490	2490	2584	2530	51	4904	4886	5121	4886	81	8746	8678	9049	8347
22	2295	2284	2409	2297	52	4626	4687	4910	4668	82	8666	8632	9046	8372
23	2539	2550	2650	2560	53	4690	4667	4976	4666	83	8699	8650	9118	8265
24	2409	2430	2533	2399	54	4714	4742	4958	4650	84	8678	8725	9105	8363
25	2494	2492	2610	2538	55	4653	4662	4975	4475	85	8700	8679	9011	8304
26	2441	2458	2532	2467	56	4739	4705	4992	4521	86	8797	8709	9081	8450
27	2450	2445	2557	2502	57	4709	4675	4986	4576	87	8850	8763	9174	8507
28	2407	2380	2508	2411	58	4778	4750	4964	4688	88	8866	8834	9174	8584
29	2440	2426	2574	2521	59	4721	4685	4960	4532	89	8723	8719	9067	8341
30	2406	2369	2519	2407	60	4785	4779	4993	4846	90	8870	8856	9266	8489

Tabulka 12: Porovnání nejlepších výsledků algoritmů s implementovaným lokálním vyhledáváním.

JxM	ABC_l	DDE_l	HS_l	RON	JxM	ABC_l	DDE_l	HS_l	RON	JxM	ABC_l	DDE_l	HS_l	RON
20x5					50x5					100x5				
1	1377	1374	1374	1384	31	3112	3076	3100	3151	61	6354	6352	6373	6455
2	1414	1411	1411	1411	32	3274	3266	3272	3395	62	6242	6195	6199	6214
3	1281	1289	1280	1294	33	3111	3073	3080	3184	63	6127	6099	6119	6124
4	1449	1448	1448	1448	34	3209	3198	3212	3303	64	5965	5936	5929	5976
5	1342	1344	1344	1366	35	3252	3232	3232	3272	65	6187	6141	6174	6173
6	1363	1363	1363	1363	36	3259	3244	3240	3400	66	6082	6038	6027	6094
7	1388	1381	1381	1381	37	3107	3092	3100	3228	67	6224	6186	6206	6262
8	1384	1379	1379	1384	38	3135	3105	3138	3260	68	6144	6101	6098	6061
9	1373	1373	1373	1378	39	2990	2982	2979	3104	69	6331	6325	6306	6474
10	1283	1283	1283	1283	40	3210	3195	3206	3264	70	6373	6342	6361	6366
20x20					50x10					100x10				
11	1698	1701	1701	1698	41	3743	3729	3737	3913	71	7282	7215	7200	7496
12	1834	1833	1833	1836	42	3599	3565	3543	3798	72	7002	6943	6957	7281
13	1672	1659	1659	1677	43	3584	3559	3564	3723	73	7085	7064	7093	7400
14	1535	1535	1535	1622	44	3778	3749	3756	3885	74	7360	7307	7304	7670
15	1617	1617	1617	1658	45	3721	3690	3694	3934	75	7052	7009	7005	7317
16	1604	1590	1592	1640	46	3701	3684	3697	3831	76	6845	6792	6855	7301
17	1622	1622	1622	1634	47	3790	3748	3776	3957	77	7010	6973	7006	7247
18	1745	1738	1731	1741	48	3655	3644	3649	3774	78	7069	7045	7046	7315
19	1751	1751	1749	1777	49	3644	3612	3617	3784	79	7249	7223	7244	7631
20	1782	1782	1782	1847	50	3714	3690	3677	3928	80	7171	7112	7158	7411

Tabulka 13: Porovnání nejlepších výsledků testovaných algoritmů s implementovaným lokálním vyhledáváním. Pokračování tabulky 12.

JxM	ABC_l	DDE_l	HS_l	RON	JxM	ABC_l	DDE_l	HS_l	RON	JxM	ABC_l	DDE_l	HS_l	RON
20x20					50x20					100x20				
21	2439	2436	2438	2530	51	4574	4579	4579	4886	81	8049	7980	8066	8347
22	2234	2234	2234	2297	52	4404	4374	4381	4668	82	8036	7985	8033	8372
23	2483	2479	2483	2560	53	4380	4381	4359	4666	83	8070	8006	8043	8265
24	2348	2348	2348	2399	54	4458	4428	4459	4650	84	8034	8004	8011	8363
25	2435	2435	2435	2538	55	4332	4336	4342	4475	85	8063	8026	8050	8304
26	2389	2388	2391	2467	56	4404	4375	4375	4521	86	8101	8073	8071	8450
27	2392	2390	2398	2502	57	4393	4382	4386	4576	87	8159	8165	8169	8507
28	2341	2328	2328	2411	58	4407	4397	4403	4688	88	8204	8192	8180	8584
29	2368	2363	2363	2521	59	4403	4390	4396	4532	89	8126	8098	8097	8341
30	2328	2323	2328	2407	60	4521	4493	4504	4846	90	8155	8145	8140	8489

13 Závěr

Cílem práce bylo zhodnocení jednotlivých algoritmů a vliv lokálního vyhledávání na jejich kvalitu produkovaných výsledků. V této práci jsme se zaměřili, na řešení problému Blocking flowshop, konkrétně na sadu problémů vytvořených Tailardem [15]. Prozkoumali jsme několik metaheurestických algoritmů a to i například implamentace částí diferenciální evoluce v artificial bee colony. Bylo prokázáno, že lokální vyhledávání má sice negativní vliv na rychlost provedení jednotlivých algoritmů, ale zároveň velmi pozitivně ovliňuje získané výsledky, kdy po implementaci lokálního vyhledávání již nezůstávali ve svých nalezených lokálních maximech, ale dokázali se přiblížit globálnímu optimu. Cíl této práce tedy splněn a předpoklad, že lokální vyhledávání zlepší schopnost prohledání možností řešení testovaných algoritmů mnohem efektivněji se prokázal jako správný.

Byl také proveden t-test viz tabulky 3 6 9, kde byl interval spolehlivosti značně nad 95% se všemi hodnotami p pod 0.05% čímž jsme prokázali že naže původní hypotéza byla zamítnuta a alternativní hypotéza byla přijata, což nám ukazuje že je zde značný rozdíl mezi implamentací s lokálním vyhledáváním a bez lokálního vyhledávání.

13.1 Možnost rozšíření

Tato práce může být dále rozšířena o další druhy metaheuristických algoritmů a vliv implementace lokálního vyhledávání. Nápříklad pro velmi populární algoritmus simulovaného žíhání nebo tabu search. Dále by mohla být práce rozšířena o výpočet všech úloh vytvořených Tailardem tedy od 20ti prací na 5ti strojích až po 500 prací na 20ti strojích. Je zde také možnost testování umístění lokálního vyhledávání do různých fází testovaných algoritmů, například pokud by bylo lokální vyhledávání implementováno na všechny fáze Artificial bee colony. Dále existuje několik druhů lokálního vyhledávání, jenž by mohli ještě dále zdokonalit získané výsledky.

14 Reference

[1] Xiaoping Lia, Qian Wanga, Cheng Wuc *Efficient composite heuristics for total flowtime minimization in permutation flow shops* Omega vol. 37,issue 1, pages 155 - 164, 2009, ISSN 0305-0483

[2] Ronconil DP *A note on constructive heuristics for the flowshop problem with blocking.* International Journal of Production Economics vol. 87,issue 1, pages 39-48, 2004, ISSN 0925-5273

[3] Grabowski J, Pempera J *The permutation flow shop problem withblocking. A tabu search approach.* Omega vol. 35,issue 3, pages 302 - 311, 2007, ISSN 0305-0483

[4] Grabowski J, Pempera J *Some local search algorithms for no-wait flow- shop problem with makespan criterion. Computers and Operations Research* Computerst and Operations Research vol 32,issue 8, pages 2197-2212, 2005, ISSN 0305-0548

[5] Abadi K, Hall N, Sriskandarajh Ch, *Minimizing cycle time in a blocking flowshop.* Operation research vol. 48, issue 1, pages 177-180, 2000, ISSN 1526-5463

[6] Caraffa V, Ianes S, Bagchi TP, Sriskandarajah C. *Minimizing makespan in a blocking flowshop using genetic algorithms.* International Journal of Production Economics vol. 70,issue 2, pages 101-115, 2001, ISBN 0925-5273

[7] Storn R, Price K *Differential evolution—a simple and efficient adaptive scheme for global optimization over continuous spaces.* In Technical Report TR-95-012, 1995

[8] Zong W. Geem, Kang S. Lee , Yongjin Park *Application of Harmony Search to Vehicle Routing.* American Journal of Applied Sciences vol. 2,issue 12, pages 1552-1557, 2005, ISSN 1546-9239

[9] Karaboga D *An idea based on honey bee swarm for numerical optimization* TECHNICAL REPORT-TR06, 2005

[10] Karaboga D, Basturk B *A powerful and efficient algorithm for numerical function optimization: artificial bee colony algorithm.* , Journal of Global Optimization vol 39, issue 3, pages 459-471, 2007, ISSN 0925-5001

[11] Singh A *An artificial bee colony algorithm for the leaf- constrained minimum spanning tree problem.* Applied Soft Computing vol. 9,issue 2, pages 625–631, 2009, ISSN 1568-4946

[12] Yu-Yan Han , Quan-Ke Pan , Jun-Qing Li , Hong-yan Sang *An improved artificial bee colony algorithm for the blocking flowshop scheduling problem* The International Journal of Advanced Manufacturing Technology vol. 60, issue 9-12 , pages 1149–1159, 2012, ISSN 0268-3768

[13] Quan-Ke Pan , M. Fatih Tasgetiren , P.N. Suganthan , T.J. Chua *A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem* Information Sciences vol. 181, Issue 12, pages 2455 - 2468, 2011, ISSN 0020-0255

[14] Q. Duan, T.W. Liao, H.Z. Yi *A comparative study of different local search application strategies in hybrid metaheuristics* Applied Soft Computing vol 13, issue 3, 2013, ISSN1464–1477

[15] Taillard E *Benchmarks for basic scheduling problems.* European Journal of Operational Research vol 64, issue 2, pages 278-285, 1993, ISSN 0377-2217

[16] Ling Wanga, Quan-Ke Panb, P.N. Suganthanc, Wen-Hong Wangb, Ya-Min Wangb *A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems* Computers and Operations Research vol. 37, issue 3, pages 509-520, 2010, ISSN 0305-0548

[17] Omran M. G. H., Mahdavi M. *Global-best harmony search.* Applied Mathematics and Computation vol 198, issue 2, pages 643-656, 2008

[18] Mahdavi M, Fesanghary M, Damangir E *An improved harmony search algorithm for solving optimization problems* Applied Mathematics and Computation vol. 188, issue 2, pages 1567–1579, 2007, ISSN 0096-3003

[19] Ling Wang, Quan-Ke Pan , Fatih Tasgetiren M *A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem.* Computers and Industrial Engineering vol 61, issue 1, pages 76-83, 2011, ISSN 0360-8352

[20] Nawaz M , Enscore E , Ham I *A heuristic algorithm for the m-machine, n-job flow shop sequencing problem.* Omega vol 11, issue 1, pages 91-95, 1983, ISSN 0305-0483

Seznam obrázků

Seznam tabulek

A Zdrojové soubory

Na přiloženém CD se ve složce „Source codes“ nachází jednotlivé C++ soubory s implementací všech testovaných algoritmů s a bez lokálního vyhledávání. jednolivé soubory jsou označeny následovně

- DE -*diferenciální evoluce*
- DE+L -*diferenciální evoluce s lokálním vyhledáváním*
- HS -*Harmony Search*
- HS + L -*Harmony search s lokálním vyhledáváním*
- ABC -*Artificial Bee Colony*
- ABC + L -*Artificial Bee Colony s lokálním vyhledáváním*

Řídící funkce jenž zajišťuje volání všech částí algoritmu je vždy nazvána „run“

B Soubory s výpočty

Na CD jsou přiloženy veškeré výstupní soubory s výpočty. Nacházejí se ve složce results. Složka je rozdělena do 3 podsložek pojmenovaných podle testovaného algoritmu. Každá z těchto složek obsahuje další dvě podsložky s názvy *results* a *results_local* obsahující textové soubory označeny *n-m* , kde n označuje tailardův test a m označuje iterací testu. n nabývá hodnot 1 až 90 a m 1 až 5. Textový soubor obsahuje následující řádky :

- Jobs *-počet prací*
- Machines - *počet storojů*
- best solution *-permutace nejlepšího nalezeného řešení*
- execution time *-doba provádění výpočtů*